OIPE JC81
JUN 1 6 2005
PATENT & TRADEMARK OFFICE

AF
IFW

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
Before the Board of Patent Appeals and Interferences

In re Patent Application of

Atty Dkt. 550-296
C# M#

CARPENTER et al

TC/A.U.: 2186

Serial No. 10/042,354

Examiner: H. Patel

Filed: January 11, 2002

Date: June 16, 2005

Title: DATA PROCESSING USING A CORPROCESSOR

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

☐ **Correspondence Address Indication Form Attached.**

☐ **NOTICE OF APPEAL**

$500.00 (1401)/$250.00 (2401) $

☒ An appeal **BRIEF** is attached in the pending appeal of the above-identified application $500.00 (1402)/$250.00 (2402) $ 500.00
Applicant hereby requests **reinstatement of the appeal** to the Board of Patent Appeals and Interferences from the last decision of the Examiner twice/finally rejecting applicant's claims(s).

☒ Credit for fees paid in prior appeal without decision on merits -$ (500.00)

☐ A reply brief is attached. **(no fee)**

☐ Petition is hereby made to extend the current due date so as to cover the filing date of this paper and attachment(s)
One Month Extension $120.00 (1251)/$60.00 (2251)
Two Month Extensions $450.00 (1252)/$225.00 (2252)
Three Month Extensions $1020.00 (1253/$510.00 (2253)
Four Month Extensions $1590.00 (1254/$795.00 (2254) $

☐ "Small entity" statement attached.

Less month extension previously paid on -$( )

TOTAL FEE ENCLOSED $ 0.00

Any future submission requiring an extension of time is hereby stated to include a petition for such time extension. The Commissioner is hereby authorized to charge any deficiency, or credit any overpayment, in the fee(s) filed, or asserted to be filed, or which should have been filed herewith (or with any paper hereafter filed in this application by this firm) to our **Account No. 14-1140.** A duplicate copy of this sheet is attached.

901 North Glebe Road, 11th Floor
Arlington, Virginia 22203-1808
Telephone: (703) 816-4000
Facsimile: (703) 816-4100
JRL:sd

NIXON & VANDERHYE P.C.
By Atty: John R. Lastova, Reg. No. 33,149

Signature: ______________

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of

CARPENTER et al — Atty. Ref.: 550-296

Serial No. 10/042,354 — Group: 2186

Filed: January 11, 2002 — Examiner: H. Patel

For: DATA PROCESSING USING A COPROCESSOR

Before the Board of Patent Appeals and Interferences

BRIEF FOR APPELLANT
On Appeal From Final Rejection
From Group Art Unit 2128

John R. Lastova
NIXON & VANDERHYE P.C.
11th Floor, 901 North Glebe Road
Arlington, Virginia 22203-1808
(703) 816-4025
Attorney for Appellant
Carpenter et al.
ARM Limited

TABLE OF CONTENTS

TABLE OF AUTHORITIES

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Patent Application of

CARPENTER et al	Atty. Ref.: 550-296
Serial No. 10/042,354	Group: 2186
Filed: January 11, 2002	Examiner: H. Patel

For: DATA PROCESSING USING A COPROCESSOR

* *

June 16, 2005

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF FOR REINSTATED APPEAL

Sir:

I. REAL PARTY IN INTEREST

The real party in interest is the assignee, ARM Limited, a corporation of the United Kingdom.

II. RELATED APPEALS AND INTERFERENCES

There are no other appeals related to this subject application. There are no interferences related to this subject application.

III. STATUS OF CLAIMS

Claims 1-7 and 9-12 are pending. The final rejection having been withdrawn after an appeal brief was filed, the Examiner withdrew the final rejection and issued another action. Claim 1 stands rejected under 35 U.S.C. §103 as being unpatentable over Gulley in view of newly-applied TI TMAA32010 User's Guide (hereafter TI), in view of Messina, and further in view of Langendorf. Claims 11 and 12 stand rejected under 35 U.S.C. §103 as being unpatentable over Gulley in view of TI and Messina. Claims 2-7 and 9 stand rejected under 35 U.S.C. §103 as being unpatentable over Gulley in view of TI, Messina, Langendorf, and further in view of York. Claim 10 stands rejected under 35 U.S.C. §103 as being unpatentable over Gulley in view of TI, Messina, Langendorf, and further in view of Wu.

IV. STATUS OF AMENDMENTS

In the advisory action dated November 12, 2004, the Examiner indicated that the after final amendment filed on November 3, 2004 would be entered for purposes of appeal.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

The focus of this application is demanding data processing environments like video image manipulation that have large amounts of data to process. One

non-limiting example is summing absolute differences for a row of pixel byte values. Figure 1 illustrates this calculation used for pixel comparisons between a current 8x8 pixel block 2 and an 8x8 reference pixel block 4.

Although special purpose digital signal processing circuitry designed to perform a restricted range of processing operations at a high speed can be used to perform specific data processing operations at high speed, such circuitry is generally inflexible. This inflexibility may mean that a relatively minor change in the processing may require expensive corresponding hardware changes. This contrasts with a general purpose processor which is designed from the outset to be able to execute a wide variety of different algorithms.

To provide both flexibility and speed, a coprocessor 10 is used with a main processor 8 to provide additional functionality not required in the basic system 6. See Figure 2. Combining coprocessor 10 with the main processor 8 advantageously increases processing throughput while still allowing the main processor 8 to accommodate different algorithms and circumstances. For example, performing a sum of absolute differences for a large quantity of data often represents a significant portion of the processing load required by a general purpose processor in carrying out operations such as pixel block matching during image processing. Off-loading the bulk of the low level calculation of the sum of absolute differences to the coprocessor 10 allows a significant increase in performance to be achieved while retaining the flexibility of the general purpose

processor 8 to employ the special purpose sum of absolute differences calculations as part of a wide variety of different algorithms.

Referring to Figure 2, a coprocessor 10 responds to a coprocessor load instruction (e.g., a USALD) executed on the main processor 8 to load one or more data words from the cache memory 12 into the coprocessor 20. The inventors discovered that speed and code density may be improved by having the coprocessor load instruction also trigger data processing operations to be performed upon operands within the loaded data words to generate result data words. Page 3, lines 22-26.

The coprocessor 10 includes a memory 18, an alignment register 20, an accumulate register 22, and a control and arithmetic function unit 24. In one example, specific coprocessor load instructions may be used to load sixteen 32-bit data words into the coprocessor memory 18. The specification provides an example on page 8, beginning at line 5 of special coprocessor load instructions (e.g., USALD instructions) being executed by the main processor 8, which then serve to load either two or three data words into the coprocessor 10. As explained beginning at line 20, the USALD instruction is passed (either directly or in the form of one or more control signals) to the coprocessor 10 where it triggers the control and arithmetic function logic 24:

> to control the loading of the required number of data words from either the cache 12 or the main memory 14 via the main processor 8 **and** then also carry out the

> sum of the absolute differences calculation using these loaded values and values from the coprocessor memory 18.

Memory systems and bus structures often only operate at certain alignments with the address base of the system. But the operand values to be manipulated by the coprocessor 10 may have a different alignment. To improve performance, the variable number of loaded data words loaded into the coprocessor 10 is advantageously determined depending upon the alignment. Consider an example where eight 8-bit operands are loaded in response to a coprocessor load instruction using word-aligned 32-bit data words. The load may be achieved either with two data words if the operands are aligned to a word boundary, or three data words if the operands are not aligned to the word boundary. This is illustrated in the right side of Figure 2.

A preferred but still example embodiment includes the coprocessor memory 18 within the coprocessor 10 to locally store data words to be used as operands in combination with the loaded data words. This is beneficial when a relatively small subset of data words are frequently required for use in combination with a much wider set of data words that are less frequently required to reduce the data channel capacity needed between the main processor and the coprocessor. Performance is also improved by passing data words to be retrieved from a memory coupled to the main processor 8 to the coprocessor 10 without them being stored within registers of the main processor 8. In this situation, the

main processor 8 functions as an address generator and memory access mechanism for the coprocessor 10.

It is particularly beneficial when the main processor 8 includes a register Rn to store an address value pointing to the data words to be loaded into the coprocessor 10. This gives control of the address pointer to the main processor 8, thereby yielding an improved degree of flexibility in the type of algorithm that may be supported. In order to enhance the role of the main processor as an address generator for the coprocessor, the coprocessor load instructions may advantageously include an offset value to be applied to the address value Rn stored as a pointer within the main processor. The offset value may optionally be used to update the pointer value either before or after the pointer value is used.

For the typical situations where alignment is generally the same for a large number of sequential accesses, an alignment register 20 within the coprocessor 10 stores an alignment value specifying the alignment between the data operands and the data words and to which the coprocessor may be responsive to control how many data words are loaded for each coprocessor load instruction. For a sum of absolute differences example application, an accumulate register 22 may advantageously be provided within the coprocessor 20 to accumulate the total sum of absolute differences calculated. Although the contents of the accumulate register 22 can be retrieved back into the main processor 8 for further manipulation, those contents are held locally within the coprocessor 10 to speed

operation and reduce the requirements for data transfer between the coprocessor and the main processor.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

The primary rejections to be reviewed on appeal are the obviousness rejections of independent claim 1 based on Gulley in view of TI, Messina, and Langendorf and of independent claims 11 and 12 based on Gulley in view of TI and Messina.

VII. ARGUMENT

A. Gully, TI, Messina, and Langendorf Fail to Teach All Features Recited in Claims 1, 11, and 12.

Gulley discloses a graphics floating point coprocessor 1200 designed to work in conjunction with a host graphic processor 120. The coprocessor 1200 performs arithmetic matrix calculations. The CPU 200 of the main graphics processor 120 is responsible for the loading of operand values to the co-processor 1200 and the retrieval of result values from the co-processor 1200. As is explained in column 11, lines 25 to 36, the main processor/CPU 200 has access to the configuration registers indicating how data operands are mapped to memory addresses. This is one reason why the loading of data values is normally regarded as a main processor operation.

Gulley fails to disclose that the coprocessor 1200 is:

> responsive to a coprocessor load instruction on said main processor to load one or more loaded data words into said coprocessor **and perform at least one coprocessor processing operation specified by said coprocessor load instruction** using said one or more loaded data words to provide operand data to generate at least one result data word.

In response to this distinction, the Examiner replies in the remarks section on pages 10-11 of the Final Office Action (and similarly in the Advisory Action):

> Gulley teaches that the coprocessor is responsive to a coprocessor load instruction (an instruction received from the host) on the main processor (the graphics processor in which the instruction comes from the host) to load one or more loaded data words (a set of operands) into the coprocessor and perform at least one coprocessor processing operation (the operation to be performed on the operands) specified by the coprocessor load instruction (the instruction received from the host) using the one or more loaded data words (the set of operands) to provide operand data to generate at least one result data word (the result) (e.g. see Col. 2, lines 3-10 and Fig. 1).

Appellants disagree.

Gulley's coprocessor is not "responsive to a coprocessor load instruction on the main processor to load one or more loaded data words <u>and perform at least one coprocessor processing operation specified by the coprocessor load instruction</u>." Gulley's coprocessor accepts "from the host a set of operands and an instruction as to the operation to be performed on the operands." Column 2, lines 7-9. As this text reveals, Gulley is describing a situation where a coprocessor receives from the

host processor one or more **other** instructions **separate from** the load instruction itself. This text does not describe the load instruction itself specifying a different coprocessor processing operation for the coprocessor to execute.

There is simply no teaching in Gulley that a load instruction used to load the operand data into the coprocessor **also** specifies the operation to be performed by the coprocessor on the loaded operand data. None of the applied supporting references relied on in the final rejection discloses this feature recited in independent claims 1, 11, and 12. Recognizing this deficiency in the references applied in the final rejection, the Examiner reopened prosecution to apply the TI reference.

The TI reference gives only very brief details of the instruction set of a DSP onto which the Examiner reads the claimed coprocessor. The Examiner refers to the LTD instruction which results in operations that can otherwise be specified by multiple separate instructions, namely LT, APAC, and DMOV. The DMOV operation falls within the I/O and Database Memory Operations section and appears to relate to the copying of the contents of one data memory location into a next data memory location. But this is not loading operand data for further processing, at least from what is described in TI.

More importantly, the Examiner ignores the fact that TI's DSP (coprocessor) is executing the DSP LTD instruction rather than the main processor as recited in the independent claims. Similarly, the claimed main processor

performs the load operation to the coprocessor (DSP) and not the coprocessor (DSP). In TI, it is the DSP (coprocessor) that executes the LTD and performs the load as well as the add and move. Clearly, the TI reference does not disclose combining into a single *main processor* instruction operations normally performed by different entities—the main processor and the coprocessor (DSP). There is no teaching in the TI reference of a main processor executing a load DSP instruction and extending that DSP load instruction to specify an operation to be performed on the information loaded onto the DSP by the main processor.

B. The Combination of Gulley and TI Is Flawed.

The Examiner contends it would have been obvious to "combine Gulley's two instructions, one for loading data words and second for performing an operation to provide result [sic], into one instruction as taught by TI." The Examiner is confusing the issue by not specifying which entity is performing which operation. TI teaches the coprocessor DSP combining a *DSP* load instruction performed by the *DSP* and another processing operation performed by the DSP on the *DSP*-loaded information into one instruction executed by the DSP. But TI does not teach a *main* processor combining a coprocessor (DSP) load instruction *executed on a main* processor with another processing operation to be performed by the DSP on the *main* processor-loaded information into one instruction executed by the DSP. Thus, neither Gulley nor TI teach this

combination of into a singl instructions operations normally handled by two different entities—a main processor and a coprocessor.

There also is no proper motivation to modify Gulley with the TI reference in such a way as to arrive at the claimed feature. Gulley's loading operations are performed by the main processor in response to one instruction, and the co-processor processing operations are performed by the co-processor in response to another instruction. There is no motivation in the TI reference that would direct a person of ordinary skill in the art to break from this typical approach exemplified by Gulley whereby the main processor is responsible for the loading of data and the co-processor is responsible for performing the co-processor processing operations. The TI reference simply teaches that the coprocessor/DSP may execute a coprocessor instruction to perform multiple DSP operations on data. A main processor is not involved and not even mentioned in the TI reference.

A further factor counsels away from making the combination of Gulley and TI as suggested by the Examiner. Producing a system in which a co-processor load instruction controls both the load and specifies the operation to be performed produces a more specialized piece of hardware with less generic applicability than would normally be considered desirable. This is discussed in the introduction of our application. The normal approach in this technical field would be to keep the load instruction separate from the instruction specifying the coprocessor processing operation to be performed to support a wider variety of applications.

This normal approach is Gulley's approach. The approach of the independent claims is contrary to the normal approach.

The Examiner asserts that the coprocessor load instruction specifying the coprocessor operation to be performed has the advantage of simplifying programming for the programmer since the programmer can add fewer sub-instructions. This assertion is not consistent with what programmers actually prefer. In practice, most programmers would find programming easier and more simplified if the load instruction and the operation specifying instruction were kept separate. One need not be a programmer to recognize that this separation of operations is conceptually easier to understand and implement and is how programs are normally constructed.

C. <u>Gulley, TI, and Messina Lack Other Features Recited in the Independent Claims.</u>

The Examiner admits that Gulley also fails to teach that the number of loaded data words loaded into the coprocessor 1200 depends on whether the start address of the operand data is aligned with a word boundary. The Examiner contends that Messina discloses the claim feature that "in response to said coprocessor load instruction," the coprocessor loads "a variable number of loaded data words in dependence upon whether a start address of said operand data within said one or more loaded data words is aligned with a word boundary." The Examiner relies on Messina's Abstract as teaching "about reading (loading) a

variable number of quad words (QWs) from the main memory." (Quoted from the Advisory Action). In the final rejection, the Examiner stated that Messina teaches "8 or 9 quad (QW) occur for a line fetch (LF) depending upon the double word (DW) boundary alignment." The Examiner's contentions do not stand up under scrutiny.

First, Messina does not relate to loading data words into a coprocessor. Messina does not even describe a coprocessor. Instead, Messina relates to the transferring data between a main memory and a cache. Neither the Examiner nor Messina provide any evidence to demonstrate that cache data transfer techniques can be applied to a coprocessor loading instruction.

Second, even if one of ordinary skill in the art were to assume the techniques described in Messina could be applied to loading data into a coprocessor, (Appellants disagree with such an assumption), Messina <u>does not load a variable number of words into a cache</u>. Although Messina *reads* a variable number of quad words (QWs) from a main memory, Messina *writes* (loads) a requested, **fixed** (not variable) number of double words (DWs) into the cache. The fixed number of requested DWs is extracted from the variable number of QWs. See column 4, line 45 to column 5, line 10. In Messina, only 8 QWs are stored into a line of the cache because this is the maximum number of QWs allowed in a cache line. If the requested, fixed number of DWs must be extracted from 9 QWs in the main memory, due to a lack of alignment of the first requested

DW with the QW boundary, then only the second DW in the first QW is stored into the cache line. Similarly, only the first DW of the final QW is stored into the cache line.

The Examiner contends this deficiency in Messina is "moot" because the Examiner alleges there is no support for the fact that Messina writes a fixed number of double words (DWs) into the cache in Messina. This sidestepping of this deficiency is remarkable for two reasons. First, if the "table is turned" back on the Examiner, the Examiner fails to provide any evidence in Messina that disproves Appellants point that Messina does not load a variable number of words into the cache. Where does Messina say that a variable number of words are loaded into the cache? The "burden of proof [is] on the Patent Office which requires it to produce the factual basis for its rejection of an application under sections 102 and 103." *In re Piasecki*, 223 USPQ 785, 788 (Fed. Cir. 1984). See also 37 CFR 1.104 (c).

Second, when Messina's teachings are read as a whole, it is clear that Messina always loads a full fixed size cache line. Messina's cache line has a fixed length: "each line having 128 bytes or 16 doublewords (DW's) of information addressed on a quadword (QW) basis." See column 3, lines 56-57. As already explained, although Messina reads double word data from a variable number of quad words, Messina does not load a variable number of quad words into the cache. In Messina's line fetch operation , the entire cache line is filled resulting in

a fixed number (16) of double words being loaded into the cache line. See abstract, Figure 9C and column 10, lines 24 to 26. The abstract describes a line fetch of 16 double words "[d]uring a line fetch (LF) of 16 DWs...[e]ither 8 or 9 QWs occur for a LF depending on the first DW boundary alignment. For a LF with 9 QWs, a write inhibit is needed for a non-data odd DW position in the last QW to avoid destroying the first DW written in the cache." Where the first DW is aligned to a QW boundary, clearly the 16 DWs of the line fetch directly correspond to 8 QWs. Where the first DW is not aligned to a QW boundary, the 16 DWs will include DWs taken from 9 QWs. Accordingly, either 8 or 9 QWs must be read. Thus, the cache in Messina loads (i.e. writes into the cache) 16 DWs (which corresponds to 8 QWs). 9 QWs cannot be written into a cache line without overwriting part of the first QW in the cache line. Messina therefore does not load a variable number of quad words into the cache, and instead, loads a fixed number (16) of double words into the cache, the 16 double words being taken from either 8 or 9 quad words. The description on column 4, line 45 to column 5, line 10 describes in detail this process of reading either 8 or 9 quad words depending on the position of a first double word in a line fetch with respect to a quad word boundary. Figure 9C illustrates the two possible relationships between the 16 double words DW0 to DW15 and either 8 (QW0 to QW7) or 9 (QW0 to QW8) corresponding quad words. It can be seen from Figure 9C that

irrespective of the relationship between the first double word and the quad word boundary, a fixed number of 16 double words is loaded into the cache.

Thus, Messina writes (loads) a requested, **fixed** (not variable) number of double words (DWs) into the cache. Accordingly, Messina, like Gulley, fails to disclose the feature that "a variable number of loaded data words are loaded into said coprocessor." This is a separate and independent basis for reversing the Examiner's rejections.

D. There Is No Legal Motivation to Combine Gulley and TI with Messina.

The Federal Circuit prohibits:

> rejecting patents solely by finding prior art corollaries for the claimed elements [because this] would permit an Examiner to use the claimed invention itself as a blueprint for piecing together elements in the prior art to defeat the patentability of the claimed invention.

In re Rouffet, 149 F.3d 1350, 1357 (Fed. Cir. 1998). Such an approach would be "an illogical and inappropriate process by which to determine patentability." *Sensonics, Inc. v. Aerosonic Corp.* 81 F.3d 1566, 1570 (Fed. Cir. 1996). Yet, this is the very approach that the Examiner is taking in an attempt to combine Messina with Gulley and the TI reference.

Messina does not load a variable number of data words into a coprocessor, but instead loads a fixed number of data words (double words) into a cache memory. In addition, Messina does not describe coprocessors. At best then,

Messina is a "prior art corollary," using the Federal Circuit's language in the *Rouffet* decision. As stated above, neither the Examiner nor Messina provide any evidence that demonstrates that cache data transfer techniques can be applied to loading instructions into a coprocessor. The Examiner's motivation to combine Gulley and Messina thus comes from the present application, and therefore, is improperly based on hindsight.

The *Rouffet* Court stated, "the Examiner must show reasons that the skilled artisan, confronted with the same problems as the inventor and no knowledge of the claimed invention, would select the elements from the prior art references for the combination in the mannered claimed." *In re Rouffet,* 149 F.3d at 1357. Gulley, by the Examiner's own admission, does not recognize the problem of operands retrieved from main processor memory not aligning with word boundaries used a coprocessor. There certainly is no recognition in Messina that the alignment could vary so that the number of number of data words loaded into the coprocessor might vary.

E. <u>Langendorf Fails to Remedy Gulley's, TI's, and Messina's Deficiencies.</u>

Independent claim 1 further recite that "said coprocessor includes an alignment register for storing a value specifying alignment between said operand data and said one or more loaded data words." The Examiner admits that neither Gulley nor Messina disclose this feature and relies on Langendorf. Langendorf

relates to a branch cache system using a plurality of memory sets in which alignment bits are used to specify whether a corresponding branch target address terminates at the end of a parcel.

Langendorf's system architecture is completely different from that described in either Gulley or Messina. There is no relationship between Langendorf and Gulley except that they both involve the alignment of one data unit with another data unit. But the particular data types involved are completely different.

Because Langendorf fails to disclose a coprocessor, the alignment values of Langendorf are therefore not part of any coprocessor and are certainly not stored in an "alignment register" of a coprocessor. The alignment values in Langendorf do not specify alignment between operand data and loaded data words, but instead specify alignment between a branch instruction and a data parcel containing the end-point of the branch instruction. Accordingly, the alignment values do not serve as a trigger to specify a variable number of data words to be loaded into a coprocessor, as is the case in the independent claims. For situations where alignment is generally the same for a large number of sequential accesses, having an alignment register within the coprocessor store an alignment value specifying the alignment between the data operands and the data words allows the coprocessor to control how many data words are loaded for each coprocessor load instruction. Ultimately, this improves transfer efficiency.

In addition, the teachings of Langendorf are also irrelevant because Messina ***already describes an indication of alignment*** using the DW address bit 28. Consequently, there would have been no motivation to modify Messina with Langendorf because Langendorf's teachings do not add to or improve upon Messina's teachings.

F. The Examiner Admits That An Alignment Register Is Not Disclosed in Gulley, TI, or Messina.

On page 4 of the Office action, the Examiner admits: "[n]one of Gulley, TI or Messina teaches the further limitation of having an alignment register for storing a value specifying alignment between the operand data and the one or more loaded data words." Yet, independent claims 11 and 12 recite loading: "into said coprocessor in dependence upon a value stored in an *alignment register* within said coprocessor, said value indicating whether a start address of said operand data within said one or more loaded data words is aligned with a word boundary." The outstanding rejection of claims 11 and 12 on page 5 of the Office action only recites Gulley, TI, and Messina. The rejection as stated cannot be sustained. Moreover, if the Examiner meant to include Langendorf to reject claims 11 and 12, then the deficiencies of such a rejection that includes Langendorf as a fourth reference have already been set forth with respect to claim 1.

VIII. CONCLUSION

Even using four references in an attempt to find all of the features recited in independent claim 1 and three (or four) references for claims 11 and 12, the Examiner has been unsuccessful. There are multiple features of the independent claims not disclosed or suggested by the applied patents. Combining so many pieces of prior art which do not inter-relate is indicative of the Examiner's excessive reliance on hindsight in analyzing the patentability of the pending claims. In addition to using three/four references to reject the independent claims, there is no proper motivation (for the reasons explained above) for combining the references of Gulley, TI, Messina, and Langendorf. Appellants have presented multiple independent bases that require reversal of the Examiner's rejections.

Respectfully submitted,

NIXON & VANDERHYE P.C.

By: ______________________
John R. Lastova
Reg. No. 33,149

JRL/kmm
Enclosures
Appendix A - Claims on Appeal

IX. CLAIMS APPENDIX

1. Data processing apparatus comprising:

(i) a main processor responsive to program instructions to perform data processing operations; and

(ii) a coprocessor coupled to said main processor and responsive to a coprocessor load instruction on said main processor to load one or more loaded data words into said coprocessor and perform at least one coprocessor processing operation specified by said coprocessor load instruction using said one or more loaded data words to provide operand data to generate at least one result data word;

(iii) wherein in response to said coprocessor load instruction, said coprocessor is configured to load a variable number of loaded data words in dependence upon whether a start address of said operand data within said one or more loaded data words is aligned with a word boundary; and

(iv) wherein said coprocessor includes an alignment register for storing a value specifying alignment between said operand data and said one or more loaded data words.

2. Data processing apparatus as claimed in claim 1, wherein said coprocessor includes a coprocessor memory for storing one or more locally stored data words used as operands in said at least one coprocessor processing operation in combination with said one or more loaded data words.

3. Data processing apparatus as claimed in claim 1, comprising a memory coupled to said main processor and wherein said main processor is configured to retrieve said one or more loaded data words from said memory to said coprocessor via said main processor without being stored within registers within said main processor.

4. Data processing apparatus as claimed in claim 1, wherein said main processor includes a register operable to store an address value pointing to said one or more data words.

5. Data processing apparatus as claimed in claim 1, wherein said at least one coprocessor processing operation includes calculating a sum of absolute differences between a plurality of byte values.

6. Data processing apparatus as claimed in claim 5, wherein said coprocessor is arranged to calculate said sum of absolute differences as a sum of absolute differences between a plurality of byte values within said one or more loaded data words and corresponding ones of a plurality of byte values within said one or more locally stored data words.

7. Data processing apparatus as claimed in claim 6, wherein said coprocessor includes an accumulate register for accumulating said sum of absolute differences.

8. Data processing apparatus as claimed in claim 4, wherein said coprocessor load instruction includes an offset value to be added to said address value upon execution.

9. Data processing apparatus as claimed in claim 1, wherein said at least one coprocessor processing operation calculates a sum of absolute differences as part of block pixel value matching.

10. A method of processing data comprising the steps of:

(i) in response to program instructions, performing data processing operations in a main processor; and

(ii) in response to a coprocessor load instruction on said main processor, loading one or more loaded data words into a coprocessor coupled to said main processor and performing at least one coprocessor processing operation specified by said coprocessor load instruction using said one or more loaded data words to provide operand data to generate at least one result data word;

(iii) wherein in response to said coprocessor load instruction, a variable number of loaded data words are loaded into said coprocessor in dependence upon a value stored in an alignment register within said coprocessor, said value indicating whether a start address of said operand data within said one or more loaded data words is aligned with a word boundary.

11. A computer program product for controlling a computer to perform the steps of:

(i) in response to program instructions, performing data processing operations in a main processor; and

(ii) in response to a coprocessor load instruction on said main processor, loading one or more loaded data words into a coprocessor coupled to said main processor and performing at least one coprocessor processing operation specified by said coprocessor load instruction using said one or more loaded data words to provide operand data to generate at least one result data word;

(iii) wherein in response to said coprocessor load instruction, a variable number of loaded data words are loaded into said coprocessor in dependence upon a value stored in an alignment register within said coprocessor, said value indicating whether a start address of said operand data within said one or more loaded data words is aligned with a word boundary.

X. EVIDENCE APPENDIX

There is no evidence appendix.

XI. RELATED PROCEEDINGS APPENDIX

There is no related proceedings appendix.